

Fundamental Performance Constraints in Horizontal Fusion of In-order Cores

Pierre Salverda and Craig Zilles

Department of Computer Science
University of Illinois at Urbana-Champaign



Overview

- A sea of small in-order cores
 - very appealing from a power, thermal and clock point of view
 - ideal for TLP-rich workloads
 - can we dynamically fuse some and do well on ILP codes too?
- Ignore artifacts of fusion and distill the key issues:
 - slip-oriented execution - decoupled in-order processing
 - properties of dataflow - more chains than available ILP
- Slip-oriented execution + properties of dataflow
 - interleave chains at a few cores; OR
 - put one chain at each of many cores
- Some form of out-of-order capability is needed

Fundamentally hard

Fusion overheads



Pierre Salverda and Craig Zilles, HPCA Feb. 2008

» 2

Outline

- Background
- The state of the art
- Slip-oriented execution
- Properties of dataflow
- Smarter steering
- More slip

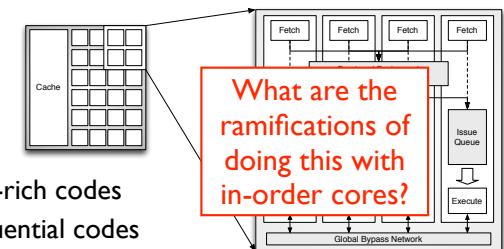


Pierre Salverda and Craig Zilles, HPCA Feb. 2008

» 3

Background

- Future workloads
 - massively parallel, TLP-rich codes
 - control-intensive, sequential codes
- The holy grail processor
 - a sea of very small, very simple cores on a single chip
 - on-demand fusion of cores to synthesize a large ILP processor
- Fusing small out-of-order cores has been proposed
 - but these are somewhat over-provisioned for TLP codes
 - and most commercial many-core designs are in-order

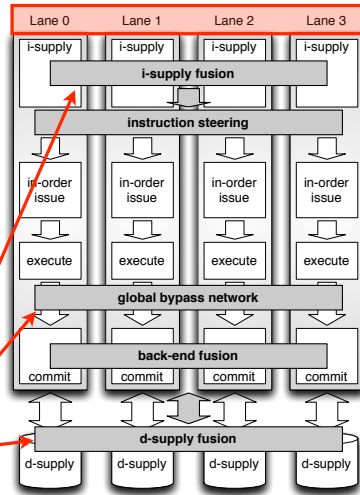


Pierre Salverda and Craig Zilles, HPCA Feb. 2008

» 4

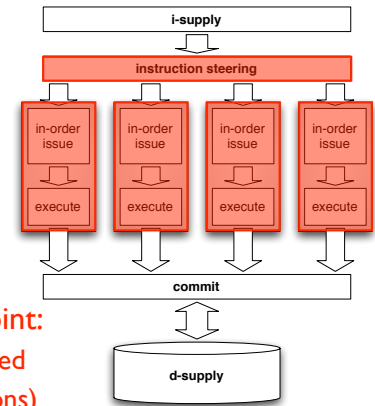
Background

- Aggregated i-supply
 - i-cache, control flow, rename, ...
 - steer instructions among lanes
- Distributed execution
 - decoupled in-order lanes
 - global communication penalty
- Aggregated d-supply
 - coherence vs banking
- Our focus is on the core
 - idealize most overheads



Background

- Idealized laned machine
 - monolithic i- and d-supply
 - zero latency steering
 - zero latency communication
- This distills the key issues
 - slip-oriented execution model
 - role played by steering
- And it facilitates our main point:
 - slip-oriented execution is limited
 - (IPC is poor despite assumptions)

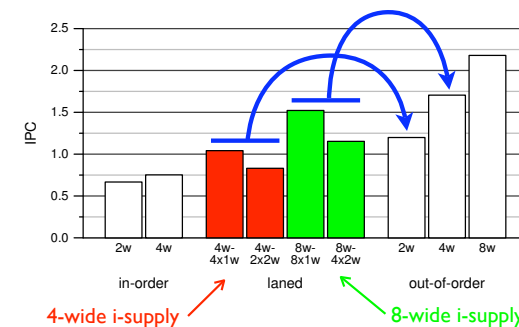


Outline

- Background
- The state of the art
- Slip-oriented execution
- Properties of dataflow
- Smarter steering
- More slip

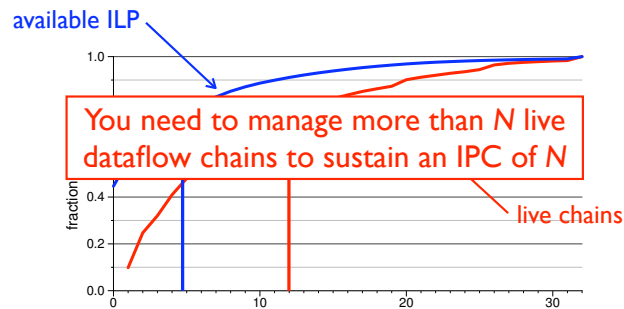
The state of the art

- Dependence-based steering [Palacharla et al, 1997]:
 - send to the lane holding a producer at its tail
 - otherwise send to an empty lane
 - otherwise stall
- Distribute as per the dataflow chains embedded in the instruction stream



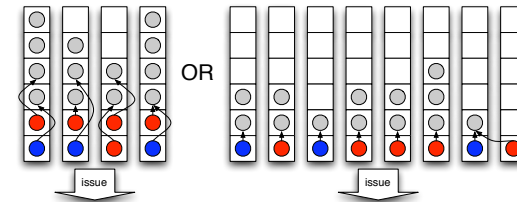
Properties of dataflow

- This is pervasive across SPEC Integer benchmarks
 - average IPC is well below the average live chain count



Properties of dataflow

- A chain must be at the head of a lane in order to issue
- To sustain an IPC of N :
 - carefully interleave $\gg N$ chains at $\sim N$ lanes; **OR**
 - have $\gg N$ lanes, one for each chain



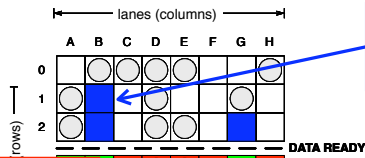
Outline

- Background
- The state of the art
- Slip-oriented execution
- Properties of dataflow
- Smarter steering
- More slip

Smarter steering

- We can reason about this abstractly
 - matrix model of per-cycle operation of lanes (see paper)
 - associate a cost with each steering decision
- Identify *fundamental* requirements for good steering:
 - constrain the delay incurred by the current instruction; **AND**
 - constrain the potential to delay subsequent instructions
- The latter is the hard one
 - must know the precise state of execution at lanes
 - incurs complexity at least as bad as dynamic scheduling

Smarter steering



A good steering policy must be cognizant of both aspects

Dataflow dependences suffice for managing this

- occupied issue slot
- empty issue slot
- empty slot obscured by later instructions
- candidate slot
- delayed beyond data ready time
- empty slot that will become obscured

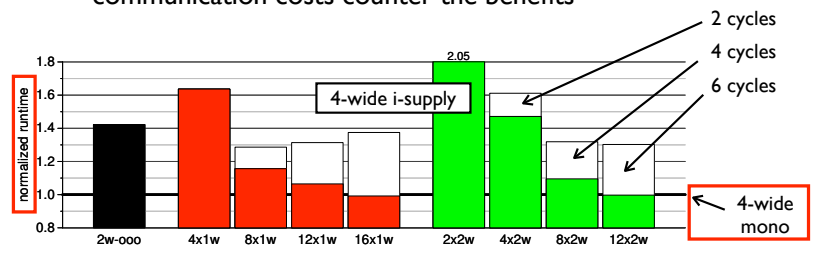
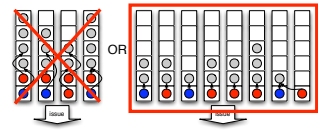
- Ultimately demands a form of prescheduling at steering
 - very sensitive to exact knowledge of execution latencies
 - inherently not amenable to pipelining

Outline

- Background
- The state of the art
- Slip-oriented execution
- Properties of dataflow
- Smarter steering
- More slip

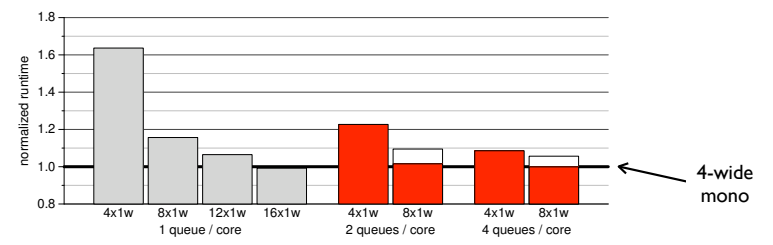
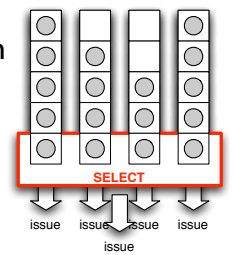
More slip

- Adding more lanes
 - each chain gets its own lane
 - simple dependence-based suffices
- As expected, performance improves
 - but lane count must be very high
 - communication costs counter the benefits



More slip

- We need issue queues, not issue bandwidth
 - share queues among execution units
 - e.g. multiple FIFO buffers per core
 - but now (simple) selection logic is needed
- Might integrate well with multithreading



Conclusion

- In-order cores are compelling for CMPs
 - power, thermal, clock benefits, and a good fit for TLP codes
 - but there's still a need for aggressive pursuit of ILP
- Fuse in-order cores
 - our focus: the underlying slip-oriented execution model
- To match monolithic IPC:
 - you need a lot of slip because dataflow has a lot of chains
 - either interleave chains at lanes or have lots of lanes
 - ... or get slip via (modest) out-of-order logic at each core
- Some form of out-of-order capability is needed